

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna informatika

Antonio Brčić

Primjena algoritama umjetne inteligencije u računalnoj igri Mlin

Završni rad

Mentor: izv. prof. dr. sc. Marina Ivašić Kos

Rijeka, 11.9.2019.

Sadržaj

1. Uvod.....	3
2. Pravila igre.....	4
3. Reinforcement learning i rule-based learning	4
4. Izrada igre.....	6
4.1 Razlike singleplayera i multiplayera.....	6
4.2 Izrada menija.....	6
4.2 Implementacija funkcija za AI – Faza 1	10
4.3 Implementacija funkcija za AI – Faza 2	18
4.3 Implementacija funkcija za AI – Faza 3	22
4.4 Implementacija završetka igre	27
4.4 Promjena postavki	29
5. Zaključak.....	31
6. Literatura i reference	32

1. Uvod

U industriji za video igre koriste se različiti algoritmi za implementaciju umjetne inteligencije sa ciljem da se igraču pruži što veća uživljenost i osjećaj realnosti u igri. Također se koristi za implementaciju različitih „uzoraka ponašanja“ u računalnoj igri. AI se pomoću danog „uzorka“ prilagođava na stil igranja igrača, a može dosizati sve od običnih protivnika do raznoraznih boss susreta. Algoritmi umjetne inteligencije se nastoje načiniti što boljima da se istovremeno prilagođavaju podacima koje otkriju te prilagođavaju igraču ovisno o postavljenoj težini i o uspješnosti igrača. Neki od poznatijih algoritama za implementaciju AI u igrama su algoritam mrava [1], algoritam pčela [2], reinforcement learning [6] i rule-based leaning [7].

U ovom radu osmišljen je jednostavan algoritam kalkulacije prioriteta koji će za tradicionalnu igru „Mlin“ [5] ili engl. Nine men's Morris, ili „Mills“ tražiti najpovoljniju poziciju figurice te najpovoljnije mjesto gdje pomaknuti istu figuricu kako bi napravio mlin, ili spriječio igrača da napravi svoj mlin. Igra je napravljena u alatu za izradu igara Unity [12].

2. Pravila igre

Igra Mlin igra se na ploči koja se sastoji od 24 pozicija horizontalno i vertikalno povezani sa 2 susjeda ako su kutne, 4 susjeda ako nisu kutne pozicije (Slika 1.). Mlin se sastoji se od 3 faze igre. U prvoj fazi započinju redom prvo plavi, zatim crveni igrač naizmjenice postavljati figurice na slobodna mjesta na ploči. Cilj je igre stvoriti Mlin, ili tri spojena susjedna mjesta, horizontalno ili vertikalno. Dijagonale se ne računaju. Prilikom postavljanja figurice ako igrač napravi mlin, on smije pritom pojesti jednu protivničku figuricu koja nije u Mlinu. Naizmjenice se postavljaju figurice dokle god oba igrača nisu svaki postavili 9 figurica. Zatim započinje druga faza igre. U drugoj fazi igre smiju se pomaknuti figurice koje imaju slobodno susjedno mjesto. U slučaju da igrač ne može pomaknuti nijednu figuricu zbog razloga što nijedna figurica nema slobodno susjedno mjesto, taj igrač gubi i igra tu završava. Igrači i u ovoj fazi pokušavaju blokirati protivničke Mlinove ili stvoriti svoje. U treću fazu ulazi igrač koji je ostao na samo 3 figurice. Kada jedan od igrača ostane na 3 figurice, oni smiju pomaknuti bilo koju svoju figuricu bilo gdje na ploči, tj. nisu limitirani na samo pomicanje figurice na susjedne pozicije. U slučaju da drugi igrač u to vrijeme ostane na 3 figurice, i on ulazi u treću fazu. Igra završava kada jedan od igrača ostane na 2 figurice jer tada više ne može napraviti mlin.



Slika 1 - Ploča i figurice igre Mlin

3. Reinforcement learning i rule-based learning

Reinforcement learning je područje učenja umjetne inteligencije gdje se AI agent uči kako maksimizirati rezultat u nekoj okolini. AI agent se uči kroz proces kažnjavanja i nagrađivanja određenih poteza. Također možemo odrediti jačinu kažnjavanja i nagrađivanja određenih poteza te tako postizati optimalnije poteze za AI agenta. Na taj način možemo modelirati različitu težinu igre. Npr. u easy modeu će AI odabirati nekakav nasumični prioritet, dokle u medium i hard mode se služimo algoritmima za dodjeljivanje prioriteta. Cilj je da AI agent napravi najoptimalniji potez

u bilo kojoj danoj situaciji. Sub-optimalni potezi ne moraju se nužno kažnjavati već im se mijenja jačina prioriteta. Fokus je na otkrivanje ravnoteže između istraživanja i iskorištavanja trenutnog znanja algoritma.

Rule-based machine learning algoritam zadaje AI-u pravila pomoću kojih on identificira, uči, evolviraju, manipulira, sprema i primjenjuje podatke. Definirajuća karakteristika rule-based learninga je prepoznavanje i iskorištavanje zadanih pravila od strane korisnika. Ta se pravila uvode kako bi AI algoritmu olakšali prepoznavanje bitnih podataka te njihovu manipulaciju.

Za igru ćemo koristiti rule-based learning i reinforcement algoritam. Preko njih će AI uz Rule-based learning (if i then) ustanoviti bitne podatke i imati određeni input (potez igrača). Reinforcement learning algoritam koristimo za dodjeljivanje prioriteta pomoću kojih će AI moći odabrati optimalne linije, čvorove za pomicanje te pozicije za pomicanje kroz sve faze igre.

4. Izrada igre

4.1 Razlike singleplayera i multiplayera

Igra je izvedena u dva moda, singleplayer i multiplayer. Cilj je bio prvenstveno napraviti funkcionalnu igru u multiplayer modu te onda iz toga izvesti singleplayer. Igra se odvija identično u oba moda. U singleplayer modu korišten je identičan kod povučen iz multiplayera te je AI prilagođen tom kodu. Igrači u oba moda imaju prikazana pravila te mogu pomicati figurice i raditi mlinove u sve tri faze igre.

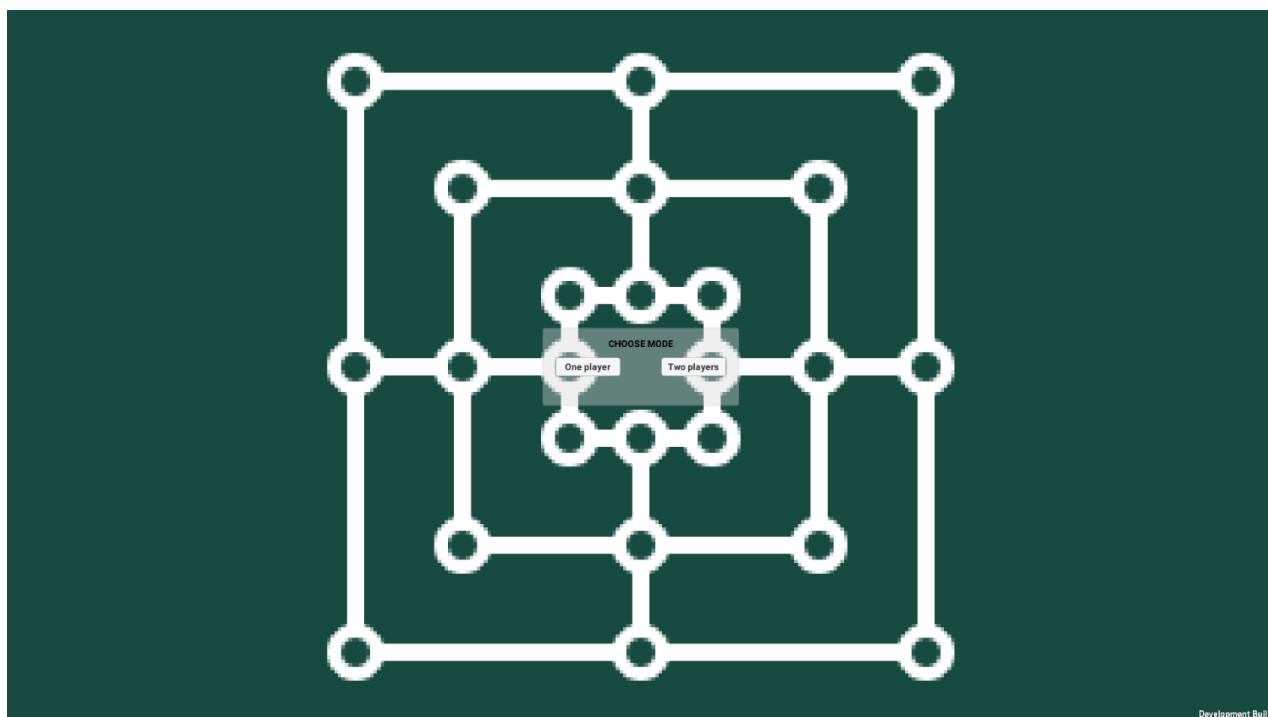
4.2 Izrada menija

Tokom izrade igre bitno je razmišljati kako implementirati algoritam umjetne inteligencije, zato je bitno unaprijed razmisliti kako će neki dio koda izgledati. Za početak igre, prikazan je meni sa opcijama play, options te quit gumbom, Slika 2. Tekst je uređen uz pomoć Unityjevog text mesh addona.



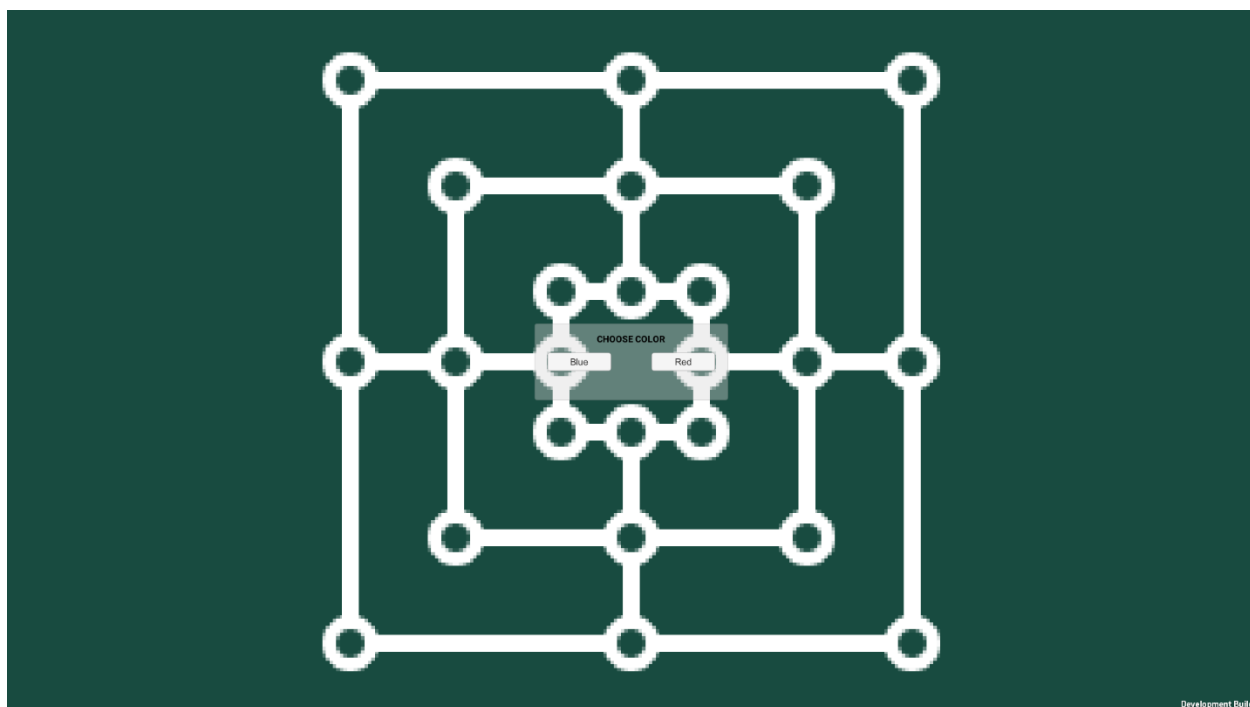
Slika 2 - Meni u igri

Klikom na play meni dolazimo do početnoga ekrana igre. Slika 3. prikazuje meni za odabir moda igranja koji je ponuđen igraču.



Slika 3 - Meni za izbor vrste igre

Igrač u One player modu može također odabrati želi li igrati prvi (blue player) ili drugi (red player) klikom na gumbove koji se prikazuju nakon što odabere one player mode na pop-up meni za biranje reda igranja, Slika 4.

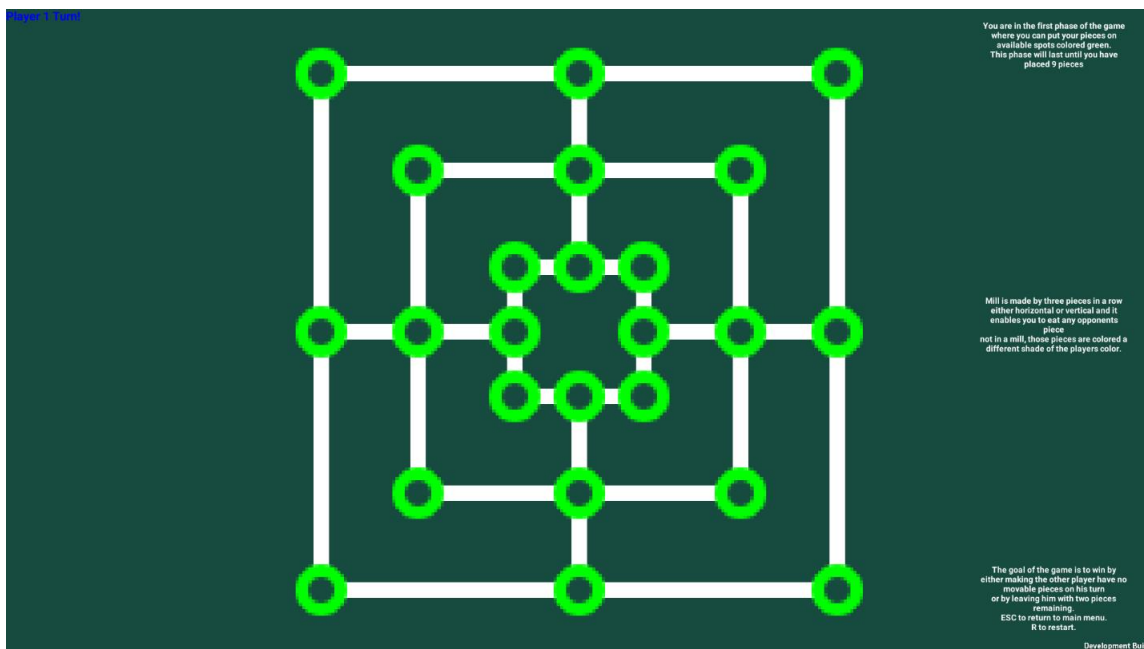


Slika 4 - Odabir reda igranja

Nakon što igrač odabere boju s kojom igra pokrene se skripta s funkcijom start koja vraća sve opcije na početne, Slika 5. Također, definirane su funkcije za pokretanje singleplayer i multiplayer mode, te funkcije za odabir reda igranja. U slučaju singleplayer moda, bez obzira koju boju igrač odabere, pokreće se StartAIMode(); funkcija koja inicira AI algoritam. Zatim se deaktivira panel za odabir i igra započinje.

```
34 void Start()
35 {
36     Options.ResetToDefault();
37 }
38
39 public void OnTwoPlayerClick()
40 {
41     choosePlayerPanel.SetActive(false);
42     StartTwoPlayerMode();
43 }
44
45 public void OnOnePlayerClick()
46 {
47     //choose player color?
48     choosePlayerPanel.SetActive(false);
49     chooseColorPanel.SetActive(true);
50 }
51
52 public void OnRedPlayerClick()
53 {
54     //start ai with red player
55     chooseColorPanel.SetActive(false);
56     Options.aiMode = true;
57     Options.aiPlayer = Player.One;
58     StartAIMode();
59 }
60
61 public void OnBluePlayerClick()
62 {
63     //start ai with blue player
64     chooseColorPanel.SetActive(false);
65     Options.aiMode = true;
66     Options.aiPlayer = Player.Two;
67     StartAIMode();
68 }
69
```

Slika 5 - Gamecontroller skripta za odabir moda



Slika 6 - Samoažurirajući tekst faza 1

Ulaskom u igru igraču se prikazuje igrača ploča sa zeleno označenim pozicijama na koje može smjestiti svoje figurice i s desne strane se objašnjavaju pravila igre. Pravila se ažuriraju ovisno o fazi igre tako da igrači znaju što im je činiti tokom svake faze ako igraju po prvi put. Slika 6. s desne strane prikazuje dinamički izmenjivi tekst koji opisuje pravila igre ovisno o fazi u kojoj se igrač nalazi. Za prikaz i izmjenu teksta pravila s desne strane ekrana koristi se funkcija `StartAIMode()`; prikazana na slici 7. Ona započinje prvu fazu igre za oba igrača i postavlja početne varijable. U slučaju kada igraju dva igrač poziva se funkcija `StartTwoPlayerMode()`; koja deaktivira AI controller skriptu te dodjeljuje obojici igrača prvu fazu. `ResetToDefault()`; postavlja sve varijable mlina na false i oslobađa sve čvorove kako bi se moglo započeti s novom igrom.

```

79
80 public void StartAIMode()
81 {
82     Debug.Log("AI mode");
83     Options.turn = Player.None;
84     Options.playerOnePhase = Phase.One;
85     Options.playerTwoPhase = Phase.One;
86     foreach (Point point in points)
87         point.owner = Player.None;
88     eat = true;
89     millChecked = true;
90     moved = true;
91     goalText.text = Options.goalText;
92     goalText.color = Color.white;
93     millText.text = Options.millText;
94     millText.color = Color.white;
95     playerPhaseText.text = Options.phaseOneText;
96     playerPhaseText.color = Color.white;
97     GameLogic();
98 }
99
100 public void StartTwoPlayerMode()
101 {
102     AiController.SetActive(false);
103     Options.turn = Player.None;
104     Options.playerOnePhase = Phase.One;
105     Options.playerTwoPhase = Phase.One;
106     foreach (Point point in points)
107         point.owner = Player.None;
108     eat = true;
109     millChecked = true;
110     moved = true;
111     goalText.text = Options.goalText;
112     goalText.color = Color.white;
113     millText.text = Options.millText;
114     millText.color = Color.white;
115     playerPhaseText.text = Options.phaseOneText;
116     playerPhaseText.color = Color.white;
117     GameLogic();
118 }
119
120 public void ResetToDefault()
121 {
122     foreach(Point point in points)
123     {
124         if (point.owner == Player.None)
125             point.NotAvailable();
126         else
127             point.NotEatable();
128         point.partOfMill = false;
129     }
130 }
131

```

Slika 7 - StartAI i StartTwoPlayer skripte

4.2 Implementacija funkcija za AI – Faza 1

Algoritmu umjetne inteligencije prvotno je potrebno dati do znanja koji su mu čvorovi susjedni u gridu i pozvati funkciju ovisno o AI fazi i modu igranja (igra prvi kao plavi, ili igra drugi kao crveni). Također je potrebno dodijeliti jednaku vrijednost (score) za svaku poziciju na ploči (Point point varijabla). Svaka pozicija ili čvor je kružić na koji se može postaviti figurica. Ima ih ukupno 24. Slika 8. prikazuje skriptu koja dodjeljuje prioritet nula svim čvorovima kako se ne bi generirale nasumične vrijednost. Varijablu lineIndexes dinamički alociramo u 2D array

[16,3] te svakoj liniji dodjeljujemo tri pointa (čvora) u njoj kako bi kasnije mogli provjeravati za mlin u tim linijama. Funkcija AILogic poziva AI za određenu fazu.

```

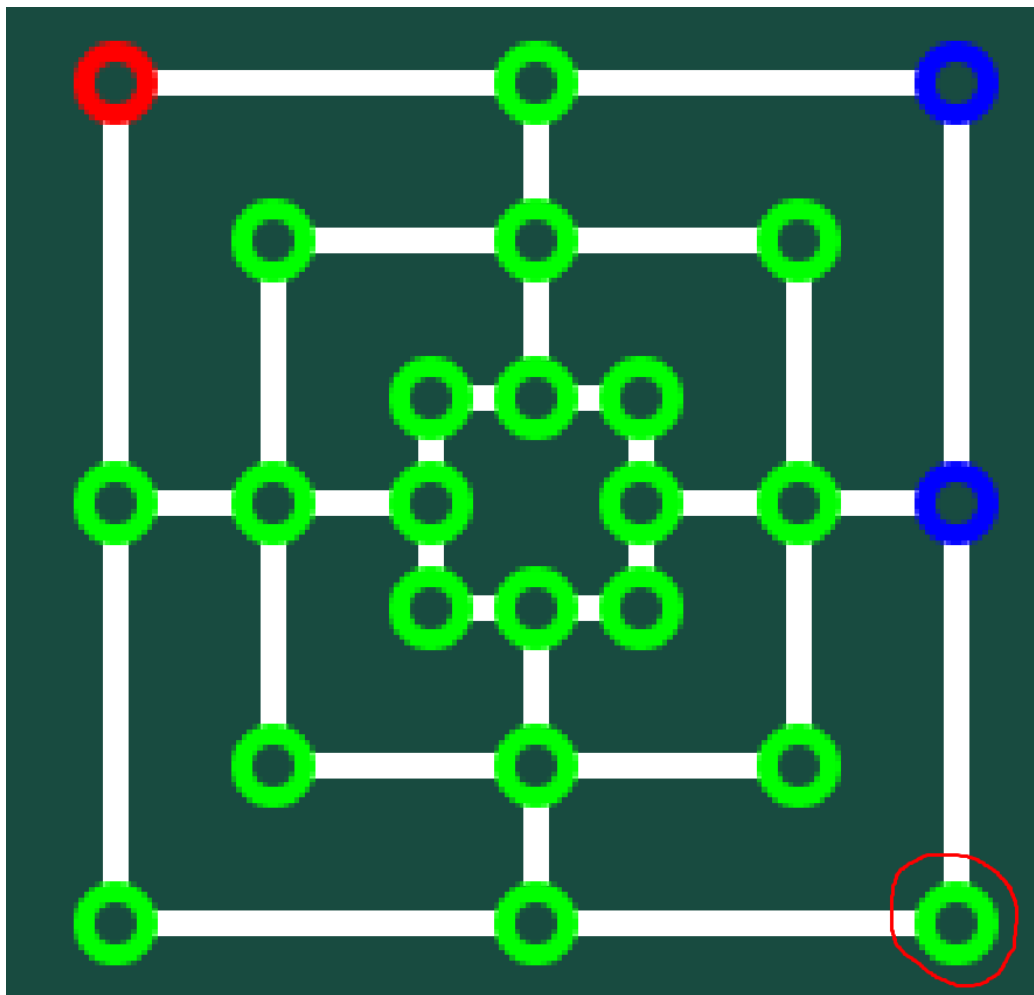
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AiController : MonoBehaviour
6  {
7      //scores for each line and point within line
8      float[,] lineScores;
9      int[,] lineIndexes = new int[16, 3] { { 0, 1, 2 }, { 0, 3, 4 }, { 2, 5, 6 }, { 4, 7, 6 }, { 8, 9, 10 }, { 8, 11, 12 }, { 10, 13, 14 },
10      { 12, 15, 14 }, { 16, 17, 18 }, { 16, 19, 20 }, { 18, 21, 22 }, { 20, 23, 22 }, { 1, 9, 17 },
11      { 3, 11, 19 }, { 5, 13, 21 }, { 7, 15, 23 } };
12
13      GameController gameController;
14
15      void Start()
16      {
17          lineScores = new float[16,3];
18          for(int i=0; i < 16; i++)
19              for (int ii = 0; ii < 3; ii++)
20                  lineScores[i,ii] = 0;
21          gameController = GameObject.FindObjectOfType<GameController>();
22      }
23
24      public void AiLogic()
25      {
26          if (gameController.eat && gameController.millExists)
27              AiLogicEat();
28          //if phase one
29          else if(Options.turn == Player.One && Options.playerOnePhase == Phase.One)
30          {
31              AiLogicFirstPhase();
32          }
33          else if(Options.turn == Player.Two && Options.playerTwoPhase == Phase.One)
34          {
35              AiLogicFirstPhase();
36          }
37          //if phase two
38          else if (Options.turn == Player.One && Options.playerOnePhase == Phase.Two)
39          {
40              AiLogicPhaseTwo();
41          }
42          else if (Options.turn == Player.Two && Options.playerTwoPhase == Phase.Two)
43          {
44              AiLogicPhaseTwo();
45          }
46          //if phase three
47          else if (Options.turn == Player.One && Options.playerOnePhase == Phase.Three)
48          {
49              AiLogicPhaseThree();
50          }
51          else if (Options.turn == Player.Two && Options.playerTwoPhase == Phase.Three)
52          {
53              AiLogicPhaseThree();
54          }
55      }
56

```

Slika 8 - AI rangiranje mjesta te pozivanja AILogic funkcija

U prvoj fazi davanja prioriteta AI prvobitno daje svakoj poziciji na ploči prioritet od 100. Zatim prebrojava koji su čvorovi slobodni i mogu se zauzeti te prebrojava koliko ima čvorova zauzetih u liniji (potrebno je imati 3 čvora u nizu u liniji za mlin). Zadnja varijabla provjerava koliko čvorova poredanih ima neprijatelj te povećava ili smanjuje score. Npr. ako AI ima dva čvora u nizu, AI će pokušati napraviti mlin tako da zauzme treći čvor u nizu (*8 povećanje), Slika 9. U slučaju kada igrač ima dva čvora u nizu, AI će mu pokušati blokirati mlin tako da zauzme treći

čvor u nizu (*7 povećanje). Protivnik ima više od nula zauzetih pozicija u toj liniji (/2 smanjenje). AI ima jedno mjesto zauzeto i 2 iduća mjesta slobodna za mlin u toj liniji (*2 povećanje). Slika 10. prikazuje funkcije za računanje prioriteta i njihove vrijednosti.



Slika 9 - Primjer za *8 povećanje

```

57 public void FirstPhaseScoring()
58 {
59     //
60     for (int i = 0; i < 16; i++)
61     {
62         //each line
63         for (int ii = 0; ii < 3; ii++)
64         {
65             //each point
66             lineScores[i, ii] = 100;
67             //number of neighbours
68             if (ii == 1)
69                 lineScores[i, ii] /= 2;
70             //number of owned points in line
71             int owned = 0;
72             if (gameController.points[lineIndexes[i, 0]].owner == Options.aiPlayer)
73                 owned++;
74             if (gameController.points[lineIndexes[i, 1]].owner == Options.aiPlayer)
75                 owned++;
76             if (gameController.points[lineIndexes[i, 2]].owner == Options.aiPlayer)
77                 owned++;
78             //number of neutral points in line
79             int neutral = 0;
80             if (gameController.points[lineIndexes[i, 0]].owner == Player.None)
81                 neutral++;
82             if (gameController.points[lineIndexes[i, 1]].owner == Player.None)
83                 neutral++;
84             if (gameController.points[lineIndexes[i, 2]].owner == Player.None)
85                 neutral++;
86             //number of enemy points in line
87             int enemy = 3 - owned - neutral;
88             if (gameController.points[lineIndexes[i, ii]].owner != Player.None)
89                 lineScores[i, ii] = 0;
90             else if (owned == 2 && neutral == 1 && gameController.points[lineIndexes[i, ii]].owner == Player.None)
91                 lineScores[i, ii] *= 8;
92             else if (enemy == 2 && neutral == 1 && gameController.points[lineIndexes[i, ii]].owner == Player.None)
93                 lineScores[i, ii] *= 7;
94             else if (enemy > 0)
95                 lineScores[i, ii] /= 2;
96             else if (owned == 1 && neutral == 2 && gameController.points[lineIndexes[i, ii]].owner == Player.None)
97                 lineScores[i, ii] *= 2;
98         }
99     }
100 }
101

```

Slika 10 – Funkcija za računanje prioriteta u prvoj fazi igre

Također u prvoj fazi AI odabire najbolju liniju za stvaranje mlina tako što daje varijabli lineScores prioritet. Kroz for petlju dodaje score najboljoj liniji. Ideja je bila pokušati simulirati situaciju gdje jedan igrač zauzme 3 kuta te onda gdje drugi igrač pokuša blokirati mlin, prvi igrač uvijek može napraviti mlin. Ako dvije linije imaju izjednačen score, odabire nasumičnu. Slika 11. prikazuje AI logika prioriteta za prvu fazu.

```

102 public void AiLogicFirstPhase()
103 {
104     List<int> bestScoreLineIndex = new List<int>();
105     FirstPhaseScoring();
106     float bestScore = 0;
107     int bestScoreIndex = 0;
108     //pick the best line
109     for (int i = 0; i < 16; i++)
110     {
111         float score = 0;
112         //each line
113         for (int ii = 0; ii < 3; ii++)
114         {
115             score += lineScores[i, ii];
116         }
117         if (score >= bestScore)
118         {
119             bestScore = score;
120             bestScoreIndex = i;
121         }
122     }
123     for (int i = 0; i < 16; i++)
124     {
125         float score = 0;
126         //each line
127         for (int ii = 0; ii < 3; ii++)
128         {
129             score += lineScores[i, ii];
130         }
131         if (score == bestScore)
132             bestScoreLineIndex.Add(i);
133     }
134     //pick the best point in line and choose it
135     int linePicked = Random.Range(0, bestScoreLineIndex.Count);
136     bestScore = 0;
137     int index = 0;
138     for (int i = 0; i < 3; i++)
139     {
140         float score = 0;
141         score += lineScores[bestScoreLineIndex[linePicked], i];
142         if (score > bestScore) {
143             bestScore = score;
144             index = i;
145         }
146     }
147     //choose the point
148     gameController.points[lineIndexes[bestScoreLineIndex[linePicked], index]].OnMouseDown();

```

Slika 11 - AI Logic u prvoj fazi

Kada AI napravi mlin, mora odabrati optimalnu neprijateljsku figuricu za pojesti. To radi u Eat scoring funkciji koristeći AIEatLogic funkciju. Bitno je napomenuti da AI ne smije pojesti figuricu koja se trenutno nalazi u mlinu. Stoga ako nema što pojesti, preskače eat fazu. Najveći prioritet ima čvor gdje protivnik ima dvije figurice poredane te mu jedna fali za mlin. Na taj način AI tjera igrača da ponovno izgubi potez postavljajući ili mičući figuricu na mjesto gdje igrač može napraviti mlin. Također AI gleda je li jedan red zauzet od protivnika, i ako nije pokušava optimalnu figuricu pomaknuti na to mjesto da napravi mlin. Tako stvara taktiku zvanu „Mill lock“ ili zaključani mlin, gdje pomakom jedne figurice igrač uvijek stvara mlin (slika 13 i 14). U kodu se

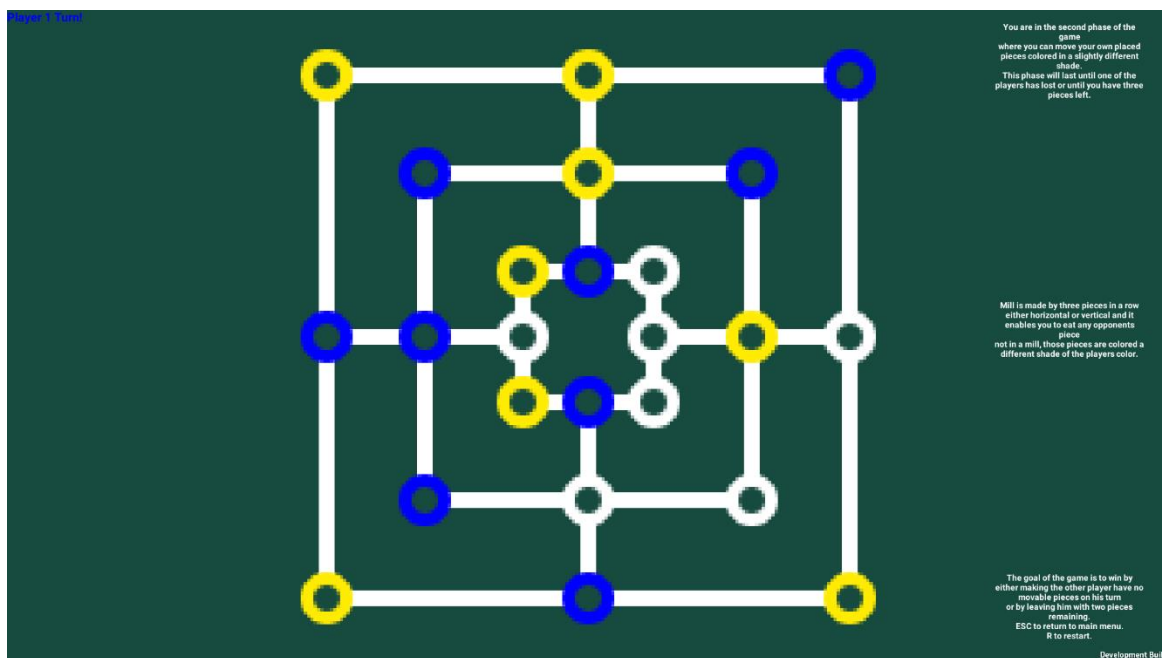
ponovno zadaju prioriteti ovisno o stanju „jestivosti“, slika 12. Ako protivnik ima dvije figurice u liniji, prioritet se diže za *5 umjesto *2.

```

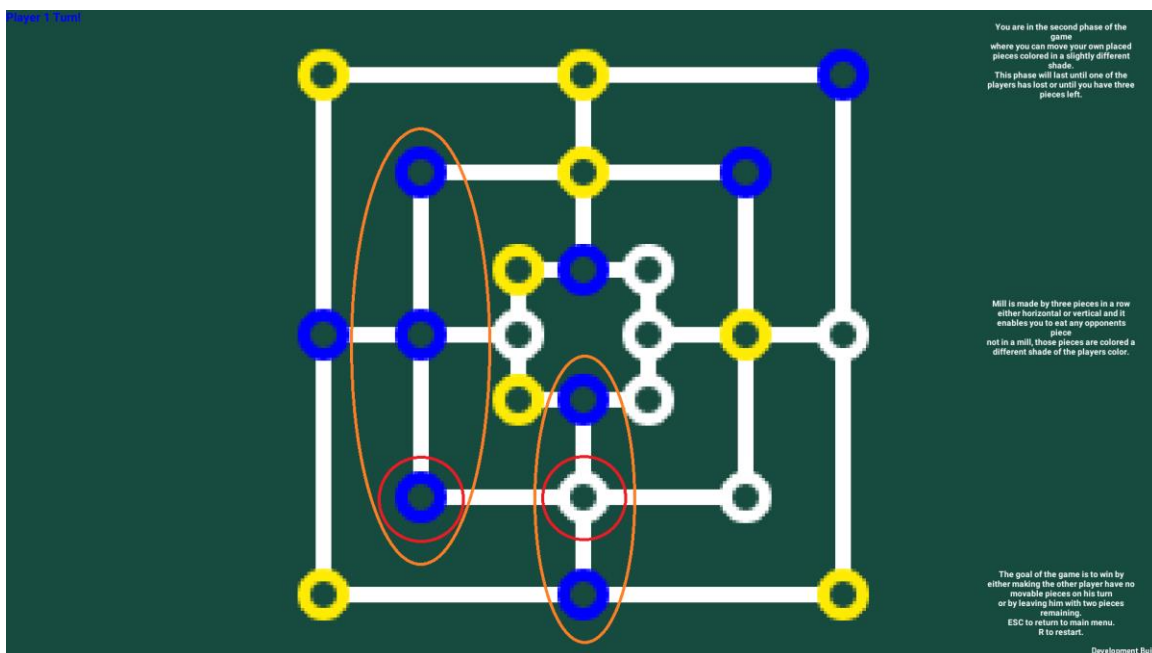
151 public void EatScoring()
152 {
153     //
154     for (int i = 0; i < 16; i++)
155     {
156         //each line
157         for (int ii = 0; ii < 3; ii++)
158         {
159             lineScores[i, ii] = 100;
160             //number of neighbours
161             if (ii == 0 || ii == 2)
162                 lineScores[i, ii] /= 2;
163             //number of owned points in line
164             int owned = 0;
165             if (gameController.points[lineIndexes[i, 0]].owner == Options.aiPlayer)
166                 owned++;
167             if (gameController.points[lineIndexes[i, 1]].owner == Options.aiPlayer)
168                 owned++;
169             if (gameController.points[lineIndexes[i, 2]].owner == Options.aiPlayer)
170                 owned++;
171             //number of neutral points in line
172             int neutral = 0;
173             if (gameController.points[lineIndexes[i, 0]].owner == Player.None)
174                 neutral++;
175             if (gameController.points[lineIndexes[i, 1]].owner == Player.None)
176                 neutral++;
177             if (gameController.points[lineIndexes[i, 2]].owner == Player.None)
178                 neutral++;
179             //number of enemy points in line
180             int enemy = 3 - owned - neutral;
181             if (gameController.points[lineIndexes[i, ii]].state != State.Eatable)
182                 lineScores[i, ii] = 0;
183             else if (enemy == 2 && gameController.points[lineIndexes[i, ii]].owner != Player.None && gameController.points[lineIndexes[i, ii]].owner != Options.aiPlayer)
184                 lineScores[i, ii] *= 5;
185             else if (enemy > 0 && gameController.points[lineIndexes[i, ii]].owner != Player.None && gameController.points[lineIndexes[i, ii]].owner != Options.aiPlayer)
186                 lineScores[i, ii] *= 2;
187         }
188     }
189 }
190

```

Slika 12 - Eat scoring

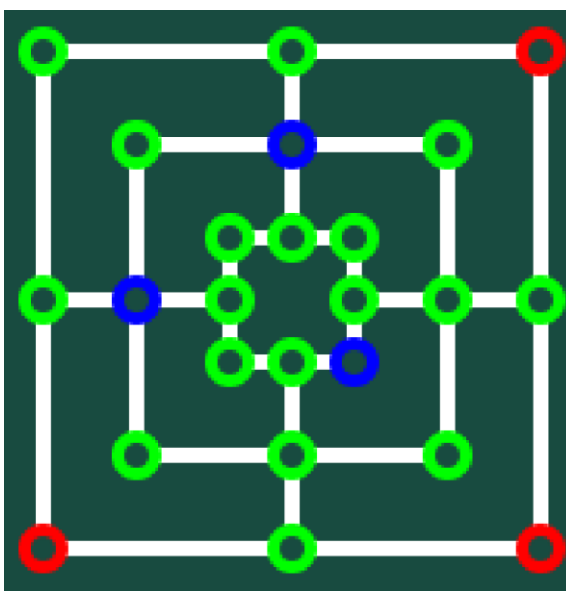


Slika 13 - Primjer zaključanog mlina



Slika 14 - Primjer zaključanog mlina s oznakama

Slika 13 i 14, prikazuju zaključan mlina plavog igrača. Pomicanjem figurice zaokružene crvenom bojom na mjesta označena crvenim plavi igrač stvara mlin u svakom potezu.



Slika 15 - AI zauzeo kuteve

U slici 15. je AI nastojao zauzeti kuteve uzevši u obzir da su linije bile slobodne. Na taj način, koju god igrač poziciju zauzeo, AI može napraviti mlin.


```

191 public void AILogicEat()
192 {
193     EatScoring();
194     float bestScore = 0;
195     int bestScoreIndex = 0;
196     //pick the best line
197     for (int i = 0; i < 16; i++)
198     {
199         float score = 0;
200         //each line
201         for (int ii = 0; ii < 3; ii++)
202         {
203             score += lineScores[i, ii];
204         }
205         if (score > bestScore)
206         {
207             bestScore = score;
208             bestScoreIndex = i;
209         }
210     }
211     //pick the best point in line and choose it
212     bestScore = 0;
213     int index = 0;
214     for (int i = 0; i < 3; i++)
215     {
216         float score = 0;
217         score += lineScores[bestScoreIndex, i];
218         if (score > bestScore)
219         {
220             bestScore = score;
221             index = i;
222         }
223     }
224     //choose the point
225     gameController.points[lineIndexes[bestScoreIndex, index]].OnMouseDown();

```

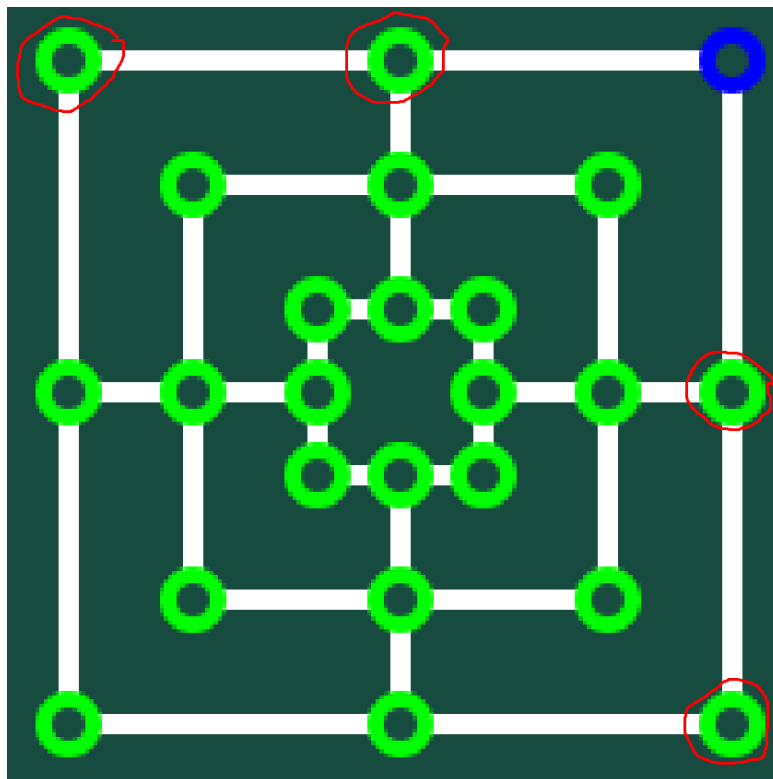
Slika 16 - AILogicEat() Funkcija

Slika 16. prikazuje AILogicEat(); funkciju koja računa prioritet specifične linije koju treba pojesti. U drugoj for petlji računa prioritet za specifično polje koje će pojesti u liniji koju je odabrao.

4.3 Implementacija funkcija za AI – Faza 2

U drugoj fazi nastaje problem. AI treba pronaći optimalnu figuricu za pomaknuti, te nakon toga naći optimalno mjesto. Najlakše je prvotno eliminirati mjesta na koja se ne može pomaknuti i figurice koje se ne mogu pomaknuti te onda tražiti prioritet za figuricu, te mjesto. Također nastaje problem pronalaska susjeda ako se nalazi u kutu, tj. takozvani problem kutnih čvorova (slika 17.).

Na slici 17. dan je primjer jednog čvora koji ima samo dva susjeda. Taj čvor sadrži informaciju samo za pola linije. Dokle ako se nalazi na horizontali ili vertikali, ima informacije za sve susjede te može provjeriti horizontalni ili vertikalni mlin. Uvodimo funkciju koja provjerava susjedni čvor od kutnog čvora te u njemu pretražuje ima li susjeda ili je li jedan od susjednih čvorova centralnog čvora zauzeti od protivnika, ali ne i oboje. Ne želimo da AI besmisleno pomiče figuricu iz kutnog čvora ako je nemoguće stvoriti mlin u toj liniji zbog toga što je jedan od čvorova zauzet od strane neprijatelja. Slike 18. i 19. prikazuju problem kutnih čvorova (slika 17.). U funkciji `isMillPossible()`; uvodimo horizontalno i vertikalno provjeravanje susjeda. Na taj način uvijek možemo znati susjedne čvorove. Provjeravanjem vertikalnog i horizontalnog mlina povećavamo sigurnost odabiranja najbolje pozicije. AI može sada provjeriti ako igrač planira napraviti horizontalni ili vertikalni mlin.



Slika 17 - Problem kutnih čvorova u igri

```

228 bool isMillPossible(Point point, Player player)
229 {
230     //NOTE: player is moving player
231     //problem: if the point is a corner, neighbours horizontal/vertical are gonna have just the middle point
232     //in other words, we will have information about half the line
233     //if the point is center, we have information about the whole line
234     //---
235     //if the point is in corner of the horiz line
236     if(point.NeighboursHorizontal.Length == 1)
237     {
238         //go one horiz neighbour further to reach the mid point and then
239         //check if the reached point is owned by enemy and one neighbour, but not both
240         //since we are moving on one of the neighbouring spots
241         Point furtherPoint = point.NeighboursHorizontal[0].GetComponent<Point>();
242         bool enemyMill = true;
243         //check for this point
244         if (furtherPoint.owner == Player.None || furtherPoint.owner == player)
245             enemyMill = false;
246         //check for one neighbour
247         bool isEnemy = false;
248         //if there's at least one enemy point in neighbours
249         foreach (GameObject go in furtherPoint.NeighboursHorizontal)
250             if (go.GetComponent<Point>().owner != Player.None && go.GetComponent<Point>().owner != player)
251                 isEnemy = true;
252         if (isEnemy == false)
253             enemyMill = false;
254         if (enemyMill == true)
255             return true;
256     }
257     else//mid
258     {
259         bool enemyMill = true;
260         //if any point in neighbours horiz from the mid point is not owned by the enemy, enemy mill is not possible
261         foreach (GameObject go in point.NeighboursHorizontal)
262             if (go.GetComponent<Point>().owner == Player.None || go.GetComponent<Point>().owner == player)
263                 enemyMill = false;
264         if (enemyMill == true)
265             return true;
266     }

```

Slika 18 - Problem kutnih čvorova

```

267 //if the point is in corner of the vert line
268 if(point.NeighboursVertical.Length == 1)
269 {
270     //go one vert neighbour further to reach the mid point and then
271     //check if the reached point is owned by enemy and one neighbour, but not both
272     //since we are moving on one of the neighbouring spots
273     Point furtherPoint = point.NeighboursVertical[0].GetComponent<Point>();
274     bool enemyMill = true;
275     //check for this point
276     if (furtherPoint.owner == Player.None || furtherPoint.owner == player)
277         enemyMill = false;
278     //check for one neighbour
279     bool isEnemy = false;
280     //if there's at least one enemy point in neighbours
281     foreach (GameObject go in furtherPoint.NeighboursVertical)
282         if (go.GetComponent<Point>().owner != Player.None && go.GetComponent<Point>().owner != player)
283             isEnemy = true;
284     if (isEnemy == false)
285         enemyMill = false;
286     if (enemyMill == true)
287         return true;
288 }
289 else//mid
290 {
291     bool enemyMill = true;
292     //if any point in neighbours vert from the mid point is not owned by the enemy, enemy mill is not possible
293     foreach (GameObject go in point.NeighboursVertical)
294         if (go.GetComponent<Point>().owner == Player.None || go.GetComponent<Point>().owner == player)
295             enemyMill = false;
296     if (enemyMill == true)
297         return true;
298 }
299 return false;
300 }
301

```

Slika 19 - Problem kutnih čvorova nastavak

U scoranju druge faze morali smo odabrati što će AI algoritmu biti bitnije. Stvarati vlastiti mlin, ili blokirati tuđi. U prvoj fazi prebrojavamo trenutno stanje kroz svaku iteraciju petlje te dodjeljujemo prioritete nanovo. Traži se optimalno mjesto i figurica za pomicanje. Zbog prijašnje implementirane funkcije, AI može lako pregledati kutne čvorove te za njih odrediti prioritet pomicanja. Stvaranje mlina mu je najveći prioritet jer tako štiti svoje figurice (ne smiju se pojesti figurice koje su u mlinu). Zatim blokiranje neprijateljskog mlina bez ugrožavanja vlastitog. U slučaju da ne može napraviti nijedno, razbija svoj mlin i priprema se vratiti istu figuricu nazad u mlin. Za slučaj da ni to ne može, pomiče nasumičnu figuru dokle god može (igra završava ako igrač ili AI ne mogu pomaknuti nijednu figuru te taj igrač gubi). Slika 20. prikazuje funkciju `PhaseTwoScoringMove()`; koja u for petlji pomoću uvjeta provjerava broj praznih čvorova te broj čvorova zauzetih od igrača ili AI-a te se pomiče ovisno o tome. Slika 21. prikazuje funkciju za zadavanje prioriteta ovisno o mogućnosti blokiranja mlina ili stvaranja mlina. Pri računanju može doći do dodjele jednakog prioriteta pomicanje figurice ili mjesta za pomicanje. Ponovo se izračunava prioritet za te pozicije te odabire više pogodna figurica i više pogodna pozicija. U slici 22. ponavljamo kod za prebrojavanje te varijablu `enemy` izračunavamo formulom $3 - \text{broj AI zauzetih polja} - \text{broj neutralnih polja}$. Na taj način dobivamo negativni koeficijent kod računanja te smanjujemo prioritet za linije i čvorove koji sadrže igračeve figurice (npr. umjesto da množimo $1 \cdot 8$ množiti ćemo $(-1) \cdot 8$ te dobiti negativni prioritet).

```

302 public void PhaseTwoScoringMove()
303 {
304     //
305     for (int i = 0; i < 16; i++)
306     {
307         //each line
308         for (int ii = 0; ii < 3; ii++)
309         {
310             lineScores[i, ii] = 100;
311             //number of owned points in line
312             int owned = 0;
313             if (gameController.points[lineIndexes[i, 0]].owner == Options.aiPlayer)
314                 owned++;
315             if (gameController.points[lineIndexes[i, 1]].owner == Options.aiPlayer)
316                 owned++;
317             if (gameController.points[lineIndexes[i, 2]].owner == Options.aiPlayer)
318                 owned++;
319             //number of neutral points in line
320             int neutral = 0;
321             if (gameController.points[lineIndexes[i, 0]].owner == Player.None)
322                 neutral++;
323             if (gameController.points[lineIndexes[i, 1]].owner == Player.None)
324                 neutral++;
325             if (gameController.points[lineIndexes[i, 2]].owner == Player.None)
326                 neutral++;
327             //number of enemy points in line
328             int enemy = 3 - owned - neutral;
329             if (gameController.points[lineIndexes[i, ii]].state != State.Movable)
330                 lineScores[i, ii] = 0;

```

Slika 20 - Scoring faza 2 part 1.

```

331     else if(gameController.points[lineIndexes[i, ii]].state == State.Movable)
332     {
333         //IF ENEMY MILL IS POSSIBLE
334         //for each neighbour from the movable point, check if it's possible to make a mill only if it's not owned
335         foreach(GameObject go in gameController.points[lineIndexes[i, ii]].NeighboursHorizontal)
336         {
337             if (go.GetComponent<Point>().owner == Player.None && isMillPossible(go.GetComponent<Point>(), Options.aiPlayer))
338                 lineScores[i, ii] *= 8;
339         }
340         //for each neighbour from the movable point, check if it's possible to make a mill only if it's not owned
341         foreach (GameObject go in gameController.points[lineIndexes[i, ii]].NeighboursVertical)
342         {
343             if (go.GetComponent<Point>().owner == Player.None && isMillPossible(go.GetComponent<Point>(), Options.aiPlayer))
344                 lineScores[i, ii] *= 8;
345         }
346         //IF OUR MILL IS POSSIBLE
347         //for each neighbour from the movable point, check if it's possible to make a mill only if it's not owned
348         foreach (GameObject go in gameController.points[lineIndexes[i, ii]].NeighboursHorizontal)
349         {
350             if (Options.aiPlayer == Player.One) {
351                 if (go.GetComponent<Point>().owner == Player.None && isMillPossible(go.GetComponent<Point>(), Player.Two) && owned != 2)
352                     lineScores[i, ii] *= 6;
353             }
354             else
355             {
356                 if (go.GetComponent<Point>().owner == Player.None && isMillPossible(go.GetComponent<Point>(), Player.One) && owned != 2)
357                     lineScores[i, ii] *= 6;
358             }
359         }
360         //for each neighbour from the movable point, check if it's possible to make a mill only if it's not owned
361         foreach (GameObject go in gameController.points[lineIndexes[i, ii]].NeighboursVertical)
362         {
363             if (Options.aiPlayer == Player.One)
364             {
365                 if (go.GetComponent<Point>().owner == Player.None && isMillPossible(go.GetComponent<Point>(), Player.Two) && owned != 2)
366                     lineScores[i, ii] *= 6;
367             }
368             else
369             {
370                 if (go.GetComponent<Point>().owner == Player.None && isMillPossible(go.GetComponent<Point>(), Player.One) && owned != 2)
371                     lineScores[i, ii] *= 6;
372             }
373         }
374     }

```

Slika 21 – Scoring faza 2 part 2.

```

375 void PhaseTwoScoringAvailable()
376 {
377     //
378     for (int i = 0; i < 16; i++)
379     {
380         //each line
381         for (int ii = 0; ii < 3; ii++)
382         {
383             lineScores[i, ii] = 100;
384             //number of owned points in line
385             int owned = 0;
386             if (gameController.points[lineIndexes[i, 0]].owner == Options.aiPlayer)
387                 owned++;
388             if (gameController.points[lineIndexes[i, 1]].owner == Options.aiPlayer)
389                 owned++;
390             if (gameController.points[lineIndexes[i, 2]].owner == Options.aiPlayer)
391                 owned++;
392             //number of neutral points in line
393             int neutral = 0;
394             if (gameController.points[lineIndexes[i, 0]].owner == Player.None)
395                 neutral++;
396             if (gameController.points[lineIndexes[i, 1]].owner == Player.None)
397                 neutral++;
398             if (gameController.points[lineIndexes[i, 2]].owner == Player.None)
399                 neutral++;
400             //number of enemy points in line
401             int enemy = 3 - owned - neutral;
402             if (gameController.points[lineIndexes[i, ii]].state != State.Available)
403                 lineScores[i, ii] = 0;
404             else if (owned == 2 && neutral == 1)
405                 lineScores[i, ii] *= 8;
406             else if (enemy == 2 && neutral == 1)
407                 lineScores[i, ii] *= 6;
408         }
409     }
410 }
411

```

Slika 22 – Scoring faza 2 part 3.

4.3 Implementacija funkcija za AI – Faza 3

U treću fazu igrač ili AI ulazi kada ostane na samo 3 figurice. Za računanje prioriteta u trećoj fazi moramo pronaći optimalnu figuricu za pomaknuti, koristeći modificirani kod za pronalazak optimalne figurice za pomicanje iz druge faze. Za računanje prioriteta pozicije koristimo dio modificiranog koda za računanje prioriteta pozicije iz prve faze te dio modificiranog koda za računanje prioriteta pomicanja ovisno o tome mogu li igrač ili protivnik napraviti mlin iz druge faze. Slika 23. prikazuje funkciju za prebrojavanje zauzetih i slobodnih polja te dodjeljivanje prioriteta i traženja optimalnog mjesta za pomicanje već od prije odabrane optimalne figurice.

```
486 public void PhaseThreeScoringMove()
487 {
488     //
489     for (int i = 0; i < 16; i++)
490     {
491         //each line
492         for (int ii = 0; ii < 3; ii++)
493         {
494             lineScores[i, ii] = 100;
495             //number of owned points in line
496             int owned = 0;
497             if (gameController.points[lineIndexes[i, 0]].owner == Options.aiPlayer)
498                 owned++;
499             if (gameController.points[lineIndexes[i, 1]].owner == Options.aiPlayer)
500                 owned++;
501             if (gameController.points[lineIndexes[i, 2]].owner == Options.aiPlayer)
502                 owned++;
503             //number of neutral points in line
504             int neutral = 0;
505             if (gameController.points[lineIndexes[i, 0]].owner == Player.None)
506                 neutral++;
507             if (gameController.points[lineIndexes[i, 1]].owner == Player.None)
508                 neutral++;
509             if (gameController.points[lineIndexes[i, 2]].owner == Player.None)
510                 neutral++;
511             //number of my points in line
512             int enemy = 3 - owned - neutral;
513             if (owned == 1 && gameController.points[lineIndexes[i, ii]].owner == Options.aiPlayer)
514                 lineScores[i, ii] *= 8;
515             else
516                 lineScores[i, ii] = 0;
517         }
518     }
519 }
```

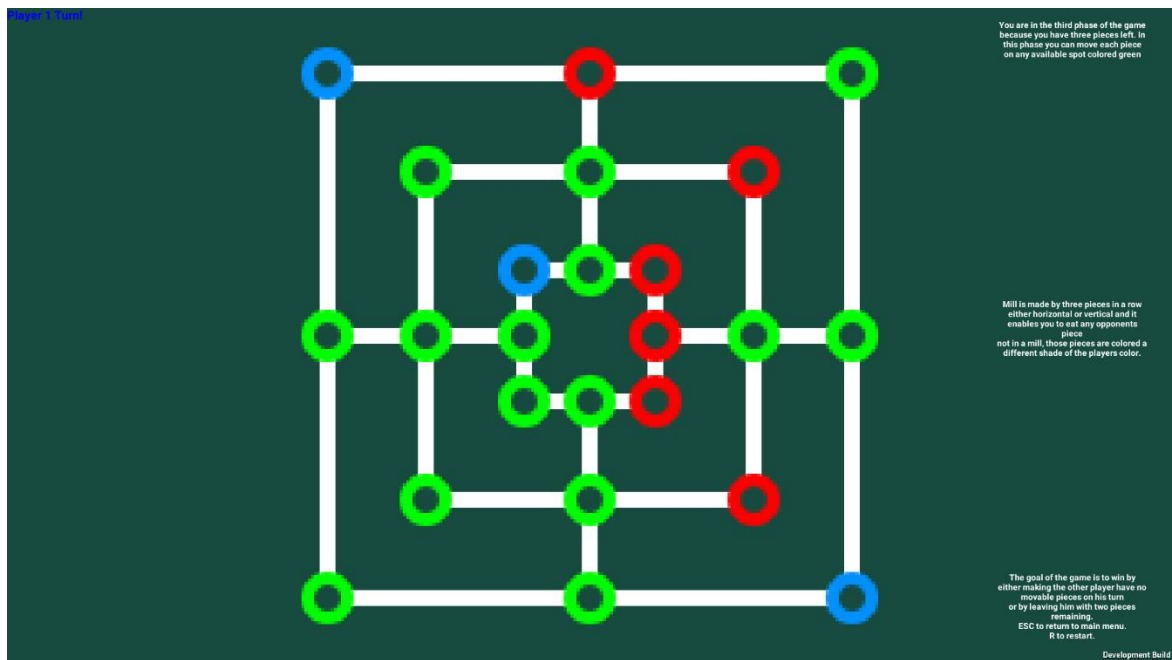
Slika 23 - AI računanje prioriteta pomicanja u fazi 3

Funkcijama su izmijenjeni koeficijenti prioriteta. AI mora paziti da ostane u mlinu kada protivnik napravi mlin, inače gubi. To provjerava kroz zadnja 2 if uvjeta. Također računa prioritet linije gdje pomaknuti figuricu i gdje da potencijalno započne stvarati mlin (slika 24).

```
521 void PhaseThreeScoreAvailable()
522 {
523     //
524     for (int i = 0; i < 16; i++)
525     {
526         //each line
527         for (int ii = 0; ii < 3; ii++)
528         {
529             lineScores[i, ii] = 100;
530             //number of owned points in line
531             int owned = 0;
532             if (gameController.points[lineIndexes[i, 0]].owner == Options.aiPlayer)
533                 owned++;
534             if (gameController.points[lineIndexes[i, 1]].owner == Options.aiPlayer)
535                 owned++;
536             if (gameController.points[lineIndexes[i, 2]].owner == Options.aiPlayer)
537                 owned++;
538             //number of neutral points in line
539             int neutral = 0;
540             if (gameController.points[lineIndexes[i, 0]].owner == Player.None)
541                 neutral++;
542             if (gameController.points[lineIndexes[i, 1]].owner == Player.None)
543                 neutral++;
544             if (gameController.points[lineIndexes[i, 2]].owner == Player.None)
545                 neutral++;
546             //number of enemy points in line
547             int enemy = 3 - owned - neutral;
548             if (enemy == 2 && neutral == 1 && gameController.points[lineIndexes[i, ii]].owner == Player.None)
549                 lineScores[i, ii] *= 8;
550             else if (owned == 2 && neutral == 1 && gameController.points[lineIndexes[i, ii]].owner == Player.None)
551                 lineScores[i, ii] *= 10;
552         }
553     }
554 }
```

Slika 24 - AI traženje slobodnih pozicija

U skripti za treću fazu AI logike, ako protivnik ne može napraviti mlin, AI ne može napraviti mlin, ili nema potrebe blokirati igračev mlin, AI traži optimalnu poziciju za stvaranje nove linije za mlin. Traži slobodnu liniju i gleda susjede zbog slučaja da ako odabere mjesto gdje ga igrač može blokirati, daje slabiji prioritet toj liniji. A pošto se u fazi tri može pomicati sa figuricama gdje hoće, mora proći kroz sve pozicije. Zato je bitan dio koda gdje traži slobodne pozicije ili pozicije zauzete od igrača. U slici 25. prikazano je kako izgleda pomicanje za plavog igrača u trećoj fazi.



Slika 25 - Faza tri za plavoga igrača

Slika 26. prikazuje implementaciju `AILogicPhaseThree()`; funkcije u koju smo implementirali već prije korišteni algoritam za računanje najbolje linije (`lineScores`). Drugi dio funkcije odabire najbolju liniju od svih ponuđenih.

```
556 public void AiLogicPhaseThree()
557 {
558     Debug.Log("phase three ai");
559     //score movable things
560     PhaseThreeScoringMove();
561     //choose movable thing
562     float bestScore = 0;
563     int bestScoreIndex = 0;
564     //pick the best line
565     for (int i = 0; i < 16; i++)
566     {
567         float score = 0;
568         //each line
569         for (int ii = 0; ii < 3; ii++)
570         {
571             score += lineScores[i, ii];
572         }
573         if (score > bestScore)
574         {
575             bestScore = score;
576             bestScoreIndex = i;
577         }
578     }
579     //pick the best point in line and choose it
580     bestScore = 0;
581     int index = 0;
582     for (int i = 0; i < 3; i++)
583     {
584         float score = 0;
585         score += lineScores[bestScoreIndex, i];
586         if (score > bestScore)
587         {
588             bestScore = score;
589             index = i;
590         }
591     }
592     //choose the point
593     gameController.points[lineIndexes[bestScoreIndex, index]].OnClick();
594     //score available things
595     PhaseThreeScoreAvailable();
596     //choose available things
597     bestScore = 0;
598     bestScoreIndex = 0;
```

Slika 26 - Traženje optimalne linije part 1

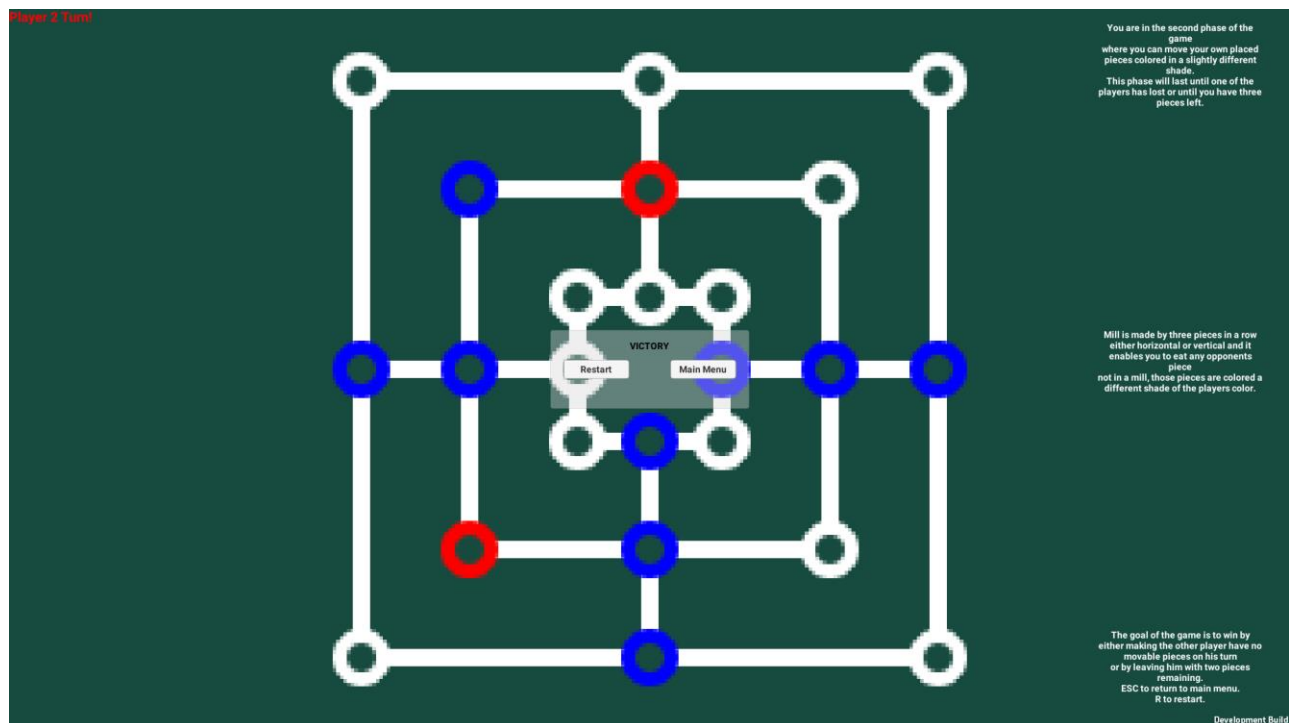
Slika 27. prikazuje da nakon odabira najbolje linije, u drugom djelu koda odabire najbolji čvor u toj liniji te njega zauzima (gameController.points).

```
599 //pick the best line
600 for (int i = 0; i < 16; i++)
601 {
602     float score = 0;
603     //each line
604     for (int ii = 0; ii < 3; ii++)
605     {
606         score += lineScores[i, ii];
607     }
608     if (score > bestScore)
609     {
610         bestScore = score;
611         bestScoreIndex = i;
612     }
613 }
614 //pick the best point in line and choose it
615 bestScore = 0;
616 index = 0;
617 for (int i = 0; i < 3; i++)
618 {
619     float score = 0;
620     score += lineScores[bestScoreIndex, i];
621     if (score > bestScore)
622     {
623         bestScore = score;
624         index = i;
625     }
626 }
627 //choose the point
628 gameController.points[lineIndexes[bestScoreIndex, index]].OnMouseDown();
629 }
630 }
631 }
```

Slika 27 - Traženje optimalne linije part 2

4.4 Implementacija završetka igre

Igra se nastavlja dok jedan igrač nema mogućnost pomicanja figurice u fazi dva, ili dokle jedan igrač ne ostane na dvije figurice u fazi tri. U glavni game controller implementirana je funkcija koja provjerava sve navedene uvjete te onda poziva end game panel meni. Igru je također moguće započeti ispočetka pritiskom tipke 'R' na tipkovnici, ili izaći iz igre pritiskom tipke 'ESC'. U end game panel meniju igrač ima opciju zaigrati ponovno ili otići u meni. Slika 28. prikazuje end game panel.



Slika 28 - Pobjeda plavog igrača, end game panel meni

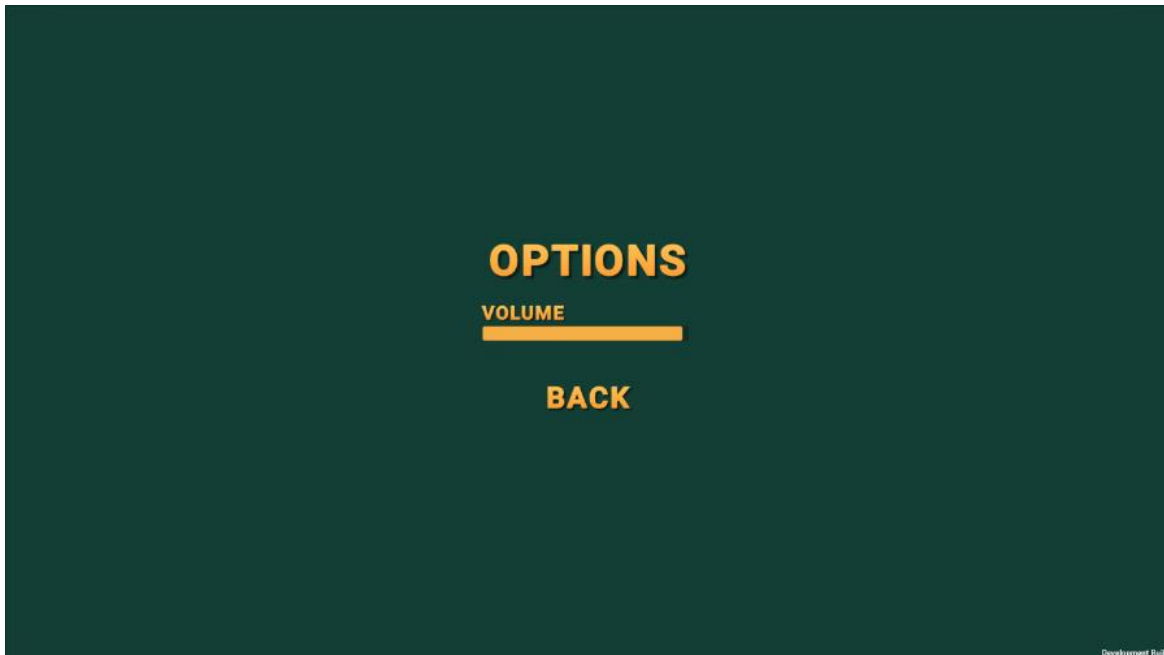
U if uvjetima se provjerava mogu li se plavi ili crveni igrač pomicati, slika 29. Također se provjerava imaju li igrači dvije figurice. Ovisno koji igrač uđe u funkciju, aktivira se victoryPanel i prikaz pobjednika.

```
159 public void CheckForDefeat()
160 {
161     //player one can't move
162     if (Options.turn == Player.One && Options.playerOneMovablePieces == 0 && Options.playerOnePhase != Phase.One) {
163         Debug.Log("Player one defeat");
164         if (Options.aiMode == true && Options.aiPlayer == Player.One)
165         {
166             victoryPanel.SetActive(true);
167             gameEnd = true;
168         }
169         else if (Options.aiMode == false)
170         {
171             victoryPanel.SetActive(true);
172             victoryTitle.text = "Red player victory";
173         }
174     }
175     else if (Options.turn == Player.Two && Options.playerTwoMovablePieces == 0 && Options.playerTwoPhase != Phase.One) {
176         Debug.Log("Player two defeat");
177         if (Options.aiMode == true && Options.aiPlayer == Player.Two)
178         {
179             victoryPanel.SetActive(true);
180             gameEnd = true;
181         }
182         else if (Options.aiMode == false)
183         {
184             victoryPanel.SetActive(true);
185             victoryTitle.text = "Blue player victory";
186         }
187     }
188     else if (Options.playerOnePiecesAmount <= 2 && Options.playerOnePhase != Phase.One) {
189         Debug.Log("Player one defeat");
190         if (Options.aiMode == true && Options.aiPlayer == Player.One)
191         {
192             victoryPanel.SetActive(true);
193             gameEnd = true;
194         }
195         else if (Options.aiMode == false)
196         {
197             victoryPanel.SetActive(true);
198             victoryTitle.text = "Red player victory";
199         }
200     }
201     else if (Options.playerTwoPiecesAmount <= 2 && Options.playerTwoPhase != Phase.One)
202     {
203         Debug.Log("Player two defeat");
204         if (Options.aiMode == true && Options.aiPlayer == Player.Two)
205         {
206             victoryPanel.SetActive(true);
207             gameEnd = true;
208         }
209         else if (Options.aiMode == false)
210         {
211             victoryPanel.SetActive(true);
212             victoryTitle.text = "Blue player victory";
213         }
214     }
215 }
216 }
```

Slika 29 - Kod za provjeru kraja igre

4.4 Promjena postavki

U igri postoji i meni sa opcijama za podešavanje zvuka. Napravljen je slider te su slider i text uređeni u TextMeshu, a kontrola zvuka implementirana je u musicController skripti. Slika 30. prikazuje gumb options koji se koristi za pojačanje i stišavanje zvuka.



Slika 30 - Options button meni

U skripti za meni (Slika 31.) definiran je slider te se on poziva na pokretanju igre. Gumb play odlazi na scenu 1, a funkcija OnVolumeChange() dozvoljava nam interaktivni slider na čiju je komponentu dodana pjesma preko music controller skripte. Slika 32. prikazuje QuitGame funkciju koja gasi igru.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using UnityEngine.UI;
6
7
8  public class MainMenuUIController : MonoBehaviour
9  {
10     public Slider volSlider;
11
12     void Start()
13     {
14         OnVolumeChange(volSlider);
15     }
16     public void PlayGame()
17     {
18         SceneManager.LoadScene(1);
19     }
20
21     public void OnVolumeChange(Slider slider)
22     {
23         Options.volume = (int)slider.value;
24         GameObject.FindObjectOfType<MusicController>().ChangeVolume();
25     }
26
27     public void QuitGame()
28     {
29         Application.Quit();
30     }
31 }
32

```

Slika 31- Meni skripta

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class MusicController : MonoBehaviour
6  {
7
8      void Start() {
9          this.gameObject.GetComponent<AudioSource>().volume = Options.volume / 100.0f;
10     }
11
12     public void ChangeVolume()
13     {
14         this.gameObject.GetComponent<AudioSource>().volume = Options.volume / 100.0f;
15     }
16 }
17

```

Slika 32 - Music controller skripta

5. Zaključak

U ovom radu bavili smo se osnovnom implementacijom algoritma umjetne inteligencije u tradicionalnu igru „Mlin“ u alatu za izradu igara i aplikacija „Unity“. Pri izradi umjetne inteligencije prvenstveno je bitno osmisliti način kako da umjetna inteligencija razmišlja kao igrač, neprijatelj i sl. Za neke stvari potrebni su algoritmi koji se prilagođavaju tome ovisno kako igrač igra. Također je moguće implementirati težine algoritma. Najjednostavniji algoritam mogao bi biti neki gdje jednostavno nasumično odabire stvari, nasumično odabire varijable i slično, te što se više povećava težina, algoritam mora više razmišljati kao igrač.

Igra je napravljena na ideji grafa. Svaka točka zna točno tko su joj susjedi, koje je boje te koje su joj susjedi boje. Tu informaciju šalje gamecontroller skripti. Na taj način lakše je pratiti promjene i vršiti pretraživanja slobodnih mjesta na bazi već predhodno poznatih informacija od susjeda.

U praktičnom djelu dana je implementacija algoritma biranja prioriteta. Naravno, idealno bi bilo da algoritam sam radi i uči, ali za dvadeset i četiri moguće pozicije i pomicanjanja na iste, te za računanja optimalnog poteza protiv igrača bilo bi potrebno po par tisuća generacija algoritma da razvije način razmišljanja za jedan čvor.

6. Literatura i reference

- [1] *Ant colony optimization algorithm*. (n.d.). Dohvaćeno iz Wikipedia:
https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms
- [2] *Bees alghoritm*. (n.d.). Dohvaćeno iz Wikipedia: https://en.wikipedia.org/wiki/Bees_algorithm
- [3] Brackeys. (n.d.). *Brackeys Youtube channel*. Preuzeto 15. 8 2019 iz Youtube:
https://www.youtube.com/watch?v=zc8ac_qUXQY
- [4] Music, H. -R. (n.d.). *Youtube*. Preuzeto 15. 8 2019 iz
<https://www.youtube.com/watch?v=8EI8ZP8LMFE>
- [5] *Pravila za igranje Mlina*. (n.d.). Dohvaćeno iz Pjesmice za Djecu:
<http://www.pjesmicezadjecu.com/drustvene-igre-pravila-igre/mlin.html>
- [6] *Reinforcement learning*. (n.d.). Dohvaćeno iz Wikipedia:
https://en.wikipedia.org/wiki/Reinforcement_learning
- [7] *Rule-Based learning*. (n.d.). Dohvaćeno iz Wikipedia: https://en.wikipedia.org/wiki/Rule-based_machine_learning
- [8] Ružić-Baf, M. (2015). *Gremo u ćap / Mettiamoci insieme nel gruppo/ Let's Flock Together Tradicionalne (odabrane) dječje igre / Giochi tradizionali (scelti) per bambini / Traditional (Selected) Children Games*. Pula: Sveučilište Jure Dobrile, Pula.
- [9] *Stack Overflow*. (15. 8 2019). Dohvaćeno iz <https://stackoverflow.com/>
- [10] Technologies, U. (n.d.). *Unity Forum*. Preuzeto 15. 8 2019 iz <https://forum.unity.com/>
- [11] Technologies, U. (n.d.). *Unity learn*. Preuzeto 15. 8 2019 iz <https://unity.com/learn>
- [12] *Unity game engine*. (n.d.). Dohvaćeno iz Wikipedia:
[https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))